

AdePT status report

Andrei Gheata, Juan Gonzalez and Witold Pokorski for the AdePT team

13.12.2023

Geant4 assessment of simulation R&D projects: AdePT and Celeritas

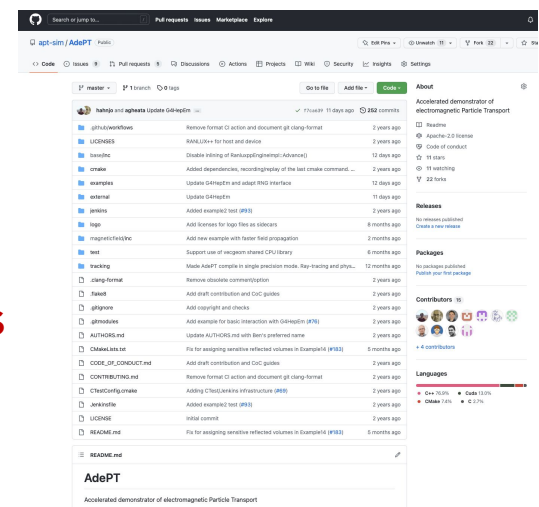


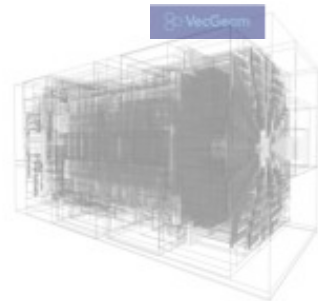
Project targets

- ▶ Understand **usability of GPUs for general particle transport simulation**, seeking for potential speed up and/or usage of available GPU resource for the HEP simulation
 - Prototype e^+ , e^- and γ EM shower simulation on GPU, evolve to realistic use-cases
- ▶ Provide GPU-friendly simulation components
 - Physics, geometry, field, but also data model and workflow
- ▶ Ensure correctness and reproducibility
 - Validate the prototype against Geant4 equivalent, ensure **reproducible results** in all modes
- ▶ Integrate in a **hybrid CPU-GPU Geant4 workflow**
 - Understand possible limitation in such an environment
- ▶ Understand **bottlenecks and blockers** limiting performance
 - Estimate feasibility and effort for efficient GPU simulation

Development approach

- ▶ **Strategy: integrate gradually features as new examples**
 - Possible at any time to create a library to link with experiment frameworks
- ▶ **Build-up gradually common functionality (services)**
 - Infrastructure: custom containers and helpers
 - Geometry: **VecGeom** library, adapting & developing for GPU
 - Physics: **G4HepEm** library, a GPU-friendly port of Geant4 EM interactions
- ▶ **Portability aspects not a major priority in this project phase**
 - Initial study identified VecGeom as blocking issue
- ▶ **Demonstrate usability in native Geant4 workflows**
 - Early integration to allow then optimizing a hybrid CPU-GPU workflow
- ▶ **Git repository**
 - Initial commit in Sep 2020, O(10) contributors





GPU geometry: VecGeom

- ▶ Built on top of the **original VecGeom GPU/CUDA** support
 - C++ types re-compiled using nvcc in a separate namespace/library
 - In AdePT we wrote a custom global navigation layer calling lower level VecGeom APIs
- ▶ Made several improvements for GPU support
 - Developed custom optimised navigation state, single-precision support
 - Moved from a simple “loop” navigator to an optimized BVH navigator
 - Adopted modern CMake GPU support
- ▶ Worked on specializing the VecGeom GPU navigation support
 - Portable, less complex code
- ▶ Added **ability to read GDML files** allowing to run with almost any geometry (essential for this R&D)
- ▶ Although being a working first solution, CSG-based approach seems to be a **bottleneck on GPU**
 - Working on **surface-based models** (see discussion later)

GPU-friendly rewrite of EM physics



- ▶ **G4HepEm**: compact library of EM processes for HEP
 - Covers the complete physics for e^- , e^+ and γ particle transport
 - Initialization of physics tables dependent on Geant4, but usage on GPU standalone and lightweight
 - Excellent physics agreement between Geant4 processes and G4HepEm

- ▶ Design of library very **supportive for heterogeneous simulations**
 - Interfaces: standalone functions without global state
 - Data: physics tables and other data structures copied to GPUs
 - Reusing > 95% of the code from G4HepEm for GPU shower simulation

GPU workflow

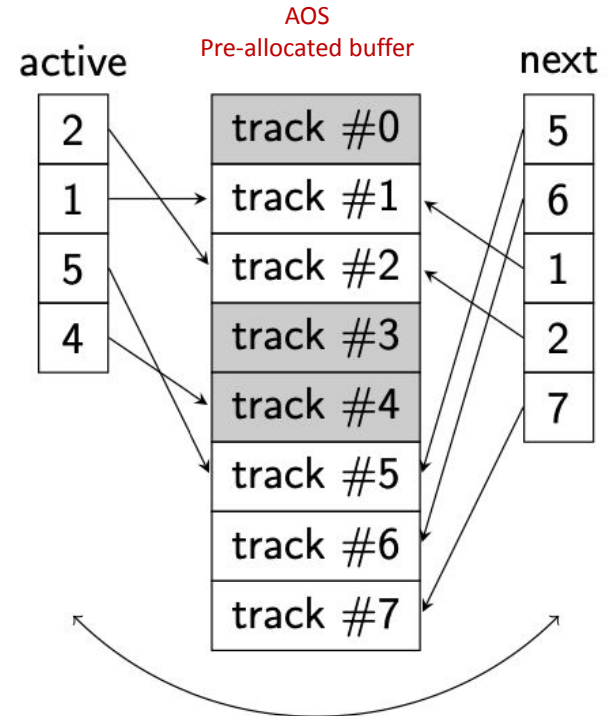
- ▶ The **GPU workflow has probably the largest impact on performance**
 - Very different on GPU compared to CPU, it has to be massively parallelizable
 - ▷ AdePT 'steps' all active tracks at once
- ▶ **Different properties** to the simulation workflow of Geant4
 - No "thread-local" state, everything associated with a track
 - At the same time: track must be as lightweight as possible
 - Data structures must not create bottlenecks (prefer atomics)
- ▶ In AdePT we adopted so far an approach based on **active track slots queues** scheduled for **per-particle kernels** (see following slides)
 - A "per-event" approach so far, easier to integrate in realistic simulation workflows

Track storage

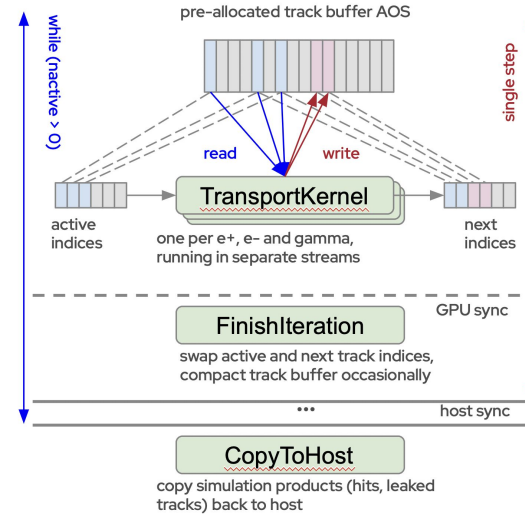
- ▶ **Properties stored per track:**
 - Random number generator state (reproducibility requirement)
 - Kinetic energy
 - Position, direction, and current navigation state
 - State to be preserved across steps (number-of-interaction-left, MSC properties)
- ▶ Pre-allocate **arrays of tracks per particle type** (array of structures)
 - One for electrons, one for positrons, one for gammas
 - Advantage: can call specialized kernels, potentially specialize stored properties
 - Atomic counter to hand out "slots" (to allow compaction)
- ▶ **Properties *not* stored per track:**
 - Particle type / PDG number (implicit from array)
 - Charge, mass (can be inferred from particle type)

Arrays of active and next tracks

- ▶ Store **indices of active tracks** (per particle type)
 - Parallelize transportation kernels over these indices
- ▶ **Queue indices for “next” active tracks**
 - Both secondaries and “surviving” tracks
 - Implemented with atomic counter
 - Tracks are killed by not enqueueing
- ▶ Run transportation kernels **stepping the active tracks**
 - Here track #1, #2 and #5 survive, track #4 dies, and track #6 and #7 are produced
- ▶ **Swap ‘active’ with ‘next’** before next iteration
 - Compacting unused slots now possible



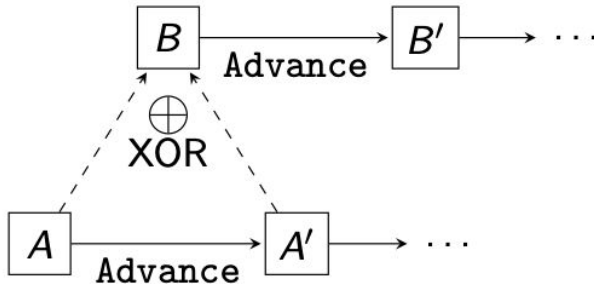
Stepping workflow



- Can start **kernels for particle types in parallel streams** (transport is independent)
- **Synchronization means overhead**
 - Synchronize with host once at the end of the step (stepping loop control)
- **Main optimization playground**
 - Better work balancing between warps, reducing impact of tails, better device occupancy
 - **Experimenting with smaller kernels** (separating discrete and continuous interactions)

Random number handling

- ▶ To assure **reproducibility**, RNG state needs to be associated with each **track**
 - Guarantees identical results no matter the parallel execution order and kernel configuration
 - Essential for debugging during development and production
- ▶ Need to initialize **new RNG state for secondary particles**
 - Must only depend on parent track to guarantee reproducibility
 - Can re-use RNG state of dying track in annihilation or conversion



Input: state A

Output: states A' and B'

1. Advance to state A'
2. XOR bits in A and A' to get B
3. Advance state B to break correlations

Random number generator: RANLUX++

- ▶ Based on the well-known **RANLUX** generator
 - Uses the equivalent LCG and therefore faster
 - Excellent statistical properties: inherited from RANLUX, only shared by MIXMAX
 - ▷ (XORWOW used by default in cuRAND known to fail some statistical tests)
- ▶ **Portable implementation available**, written with GPUs in mind
 - See J. Hahnfeld, L. Moneta: A Portable Implementation of RANLUX++
- ▶ Advantage over MIXMAX: smaller state
 - Even for $N = 17$, the default generator in Geant4 (148 bytes of state)
 - Compared to 80 bytes for RANLUX++

Magnetic Field: Runge-Kutta field propagation

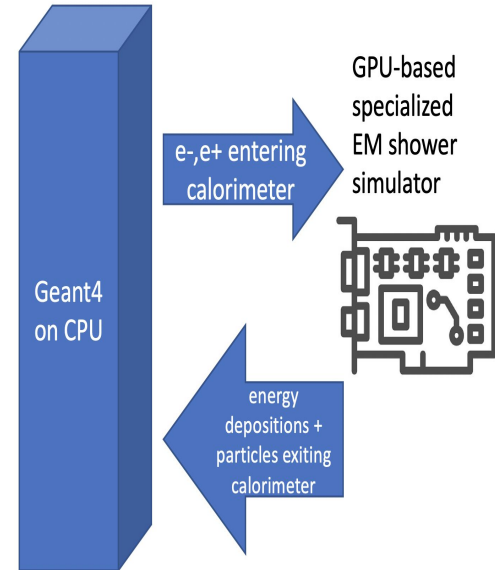
- ▶ Developed Runge-Kutta integration method has also been developed using the Dormand-Prince 5th ('DoPri5') order tableau which is the default method in Geant4
- ▶ First results with TestEm3, $B_z = 3.8\text{T}$
 - Better than per-mille agreement in observables with helix results
 - Improved handling of particles 'stuck' at boundaries - flip volume at boundary
 - Runtime about 1.5x helix (98 s vs 65 s in TestEm3 3.8T test case)
- ▶ Next steps - further testing and performance evaluation
 - Use semi-realistic field (simplified CMS or ACTS 'texture'-based interpolation)

Prototype integration strategies

- ▶ Developed two integration approaches with Geant4 application
 - 'Region-based' using fast simulation hooks
 - 'Global' using custom tracking machinery
- ▶ Both allow to delegate the simulation of EM particles to AdePT while using Geant4 for all the hadronic physics
 - AdePT as a plug-in for Geant4
- ▶ Possible to switch between full Geant4 and Geant4+AdePT configuration at run time

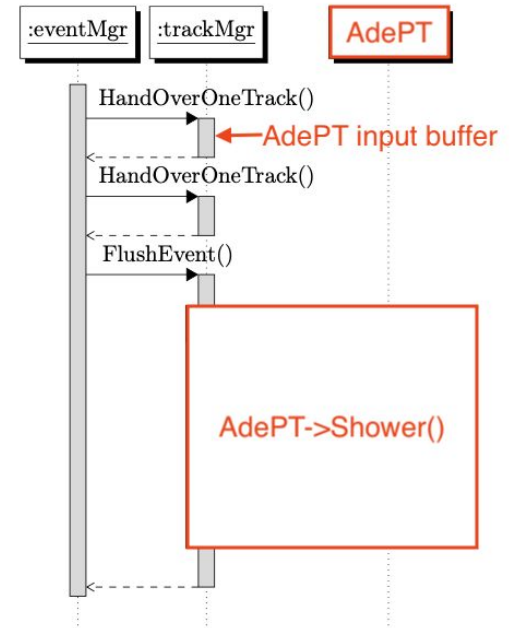
Prototype integration strategies

- ▶ **region-based approach** for delegating simulation to an external transport
 - particle **killed** on the Geant4 side **and passed** to the other transport engine
 - energy depositions and 'outgoing' (from that region) particles returned
- ▶ this follows **'fast-simulation' approach** in Geant4
 - allows the use of (most of the) existing fast-simulation hooks
 - easy integration with the physics list
 - ability to switch between full Geant4 and Geant4 + AdePT at runtime (from macro file)
- ▶ one difference:
 - we buffer **particles to process them together when some threshold is reached** (or when there are no more Geant4 particles on the stack)
 - New 'Flush' method added to Geant4 interface



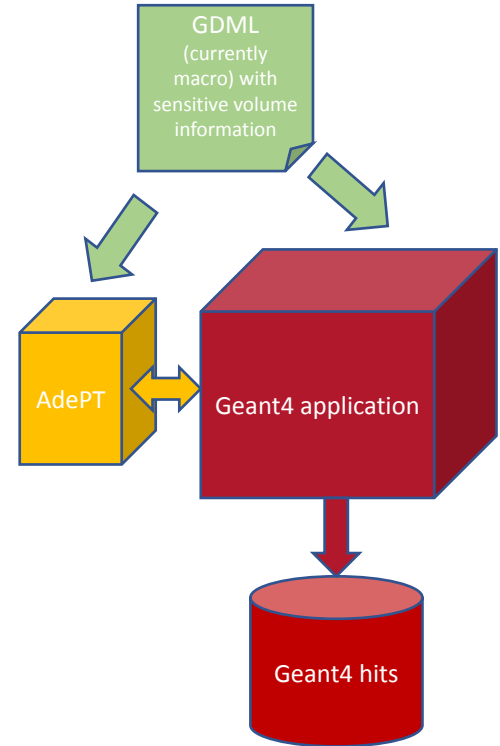
Prototype integration strategies

- ▶ Geant4 specialized tracking manager approach
 - based on the G4VTrackingManager interface
- ▶ AdePTTrackingManager class is attached to e-, e+, and gamma particle definitions inside the physics list
 - those particles not being tracked by Geant4 (anywhere), but handed over directly to AdePT
- ▶ FlushEvent method triggers the simulation on the GPU when some threshold is reached (or when the Geant4 event stack is empty)



Scoring

- ▶ **sensitive volumes marked on the GPU with a flag while initializing geometry**
 - list of sensitive volumes provided in Geant4 macro file or read from GDML auxiliary information
- ▶ **Two mechanisms of scoring implemented**
 - User scoring implemented on the device
 - 'Step' information (in sensitive volumes) transferred to the host for scoring
- ▶ **In both cases the output looks the same regardless running full Geant4 or Geant4 + AdePT**
 - can be then processed/analysed in the same way



Scoring 'on the device'

- ▶ energy deposition per volume (per event) recorded by AdePT in sensitive volumes and summed up
 - other types of 'hits' can be implemented by user
 - array of energy depositions per volume is transferred to the host once the AdePT 'shower' finished
 - indices of volumes on GPU mapped to the Geant4 ones
- ▶ Special `SensitiveDetector::ProcessHit` method (overloaded) called to translate this array of energy deposition into Geant4 hits
- ▶ Requires dedicated `ProcessHit` method
- ▶ Faster for simple scoring algorithms, but not practical for complex ones

Scoring 'on the host'

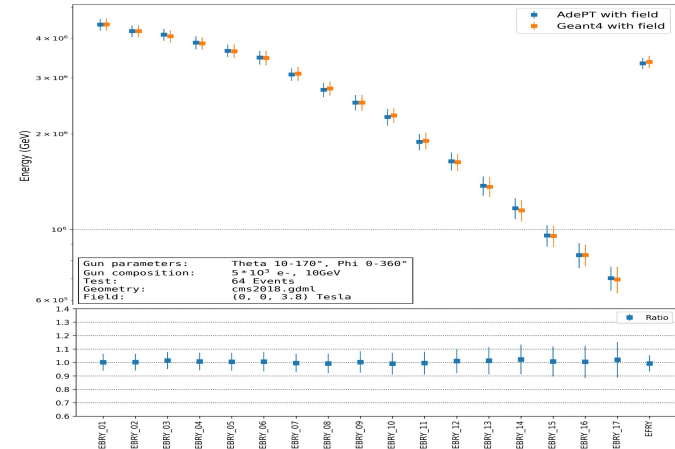
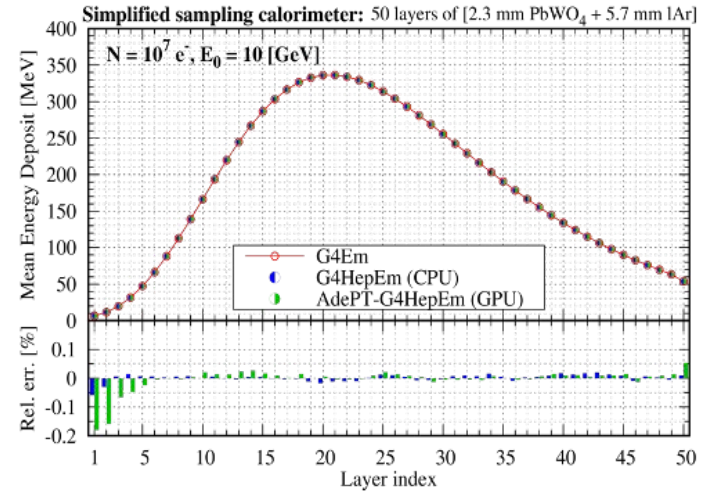
- ▶ All 'step' information (for sensitive volumes) recorded on the device and sent to the host
- ▶ G4Step objects recreated on the host
- ▶ Existing **SensitiveDetector::ProcessHits method** processes those G4Steps and created Geant4 hits
- ▶ Some (small) overhead, but allows to reuse the existing experiments scoring code for any detector

'Outgoing' particles

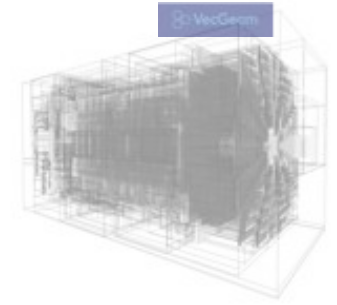
- ▶ if a particle is not to be handled by AdePT anymore (leaves AdePT region, energy goes below some threshold, etc) it is put in the **'from device' buffer**
- ▶ after AdePT shower has finished, 'from device' buffer is **transformed in Geant4 tracks** and put on the Geant4 stack
 - Geant4 continues the event loop to process those particles
 - ▷ to guarantee reproducibility (also in MT) particles 'from device' are sorted according to some unique key
- ▶ event finishes when no more particles are in the AdePT buffer ('to device') and Geant4 stacks are empty

Validation

- ▶ Validation against Geant4 standalone is essential
 - Comparisons to CPU references (in general Geant4-based) done for each added item of functionality
 - Both for standalone and Geant4 integration examples
- ▶ EM physics now fully validated
 - At $\%_0$ level in the sampling calorimeter test case
 - Both AdePT standalone as Geant4-AdePT integration showing excellent agreement



GPU support in VecGeom

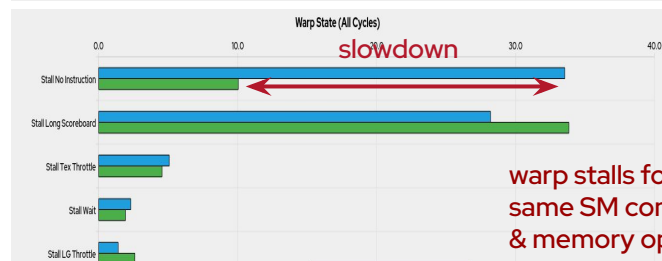
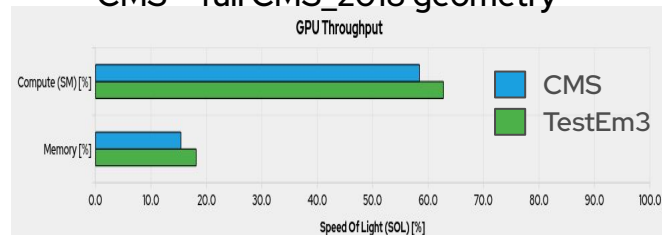


- ▶ VecGeom: backend for Geant4 navigation
 - Designed to expand the geometry modelling use cases to multi-particle workflows and GPU
 - Intended to become the only geometry modeller to have to maintain long term
- ▶ GPU support - a CUDA port of the CPU algorithms
 - Most-common Geant4 solids supported at this point
 - Work done to improve several areas in the context of the GPU simulation projects AdePT and Celeritas
 - ▷ memory footprint, robustness, performance, single precision

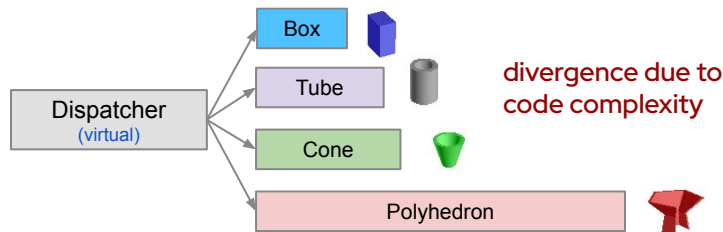
Current VecGeom solid modeling on GPU

- ▶ Several GPU unfriendly features
 - Virtual dispatch
 - Recursive code (relocation)
 - (Very) different branch complexity
- ▶ AdePT project: geometry complexity worsens performance
 - GPU performance limited by the geometry model
 - ▷ Longer stalls within warps for the same SM compute - **divergence limiting warp-level concurrency**
 - ▷ Complex register-hungry code **limiting the achievable occupancy**

TestEM3 = sampling calorimeter 50 layers
CMS = full CMS_2018 geometry

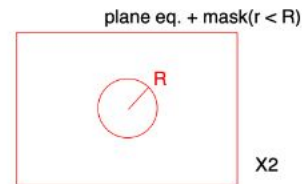
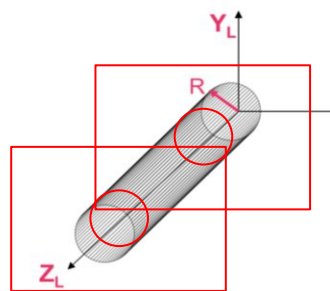
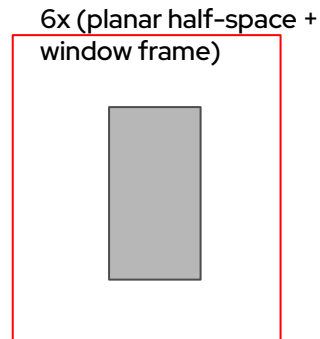
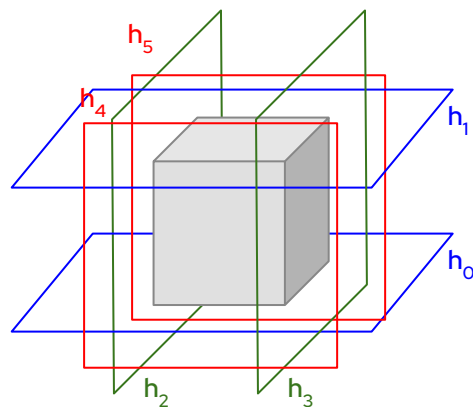


warp stalls for the same SM compute & memory ops



Bounded surface modeling

- ▶ 3D bodies represented as Boolean operation of half-spaces*
 - First and second order, infinite
 - Just intersections for convex primitives
 - ▷ e.g. box = $h_0 \& h_1 \& h_2 \& h_3 \& h_4 \& h_5$
 - Similarities with the **Orange** model
 - ▷ Evaluated Orange to start with
- ▶ Storing in addition the solid imprint (frame) on each surface: **FramedSurface**
 - Similarities with **depray** (ACTS)
 - The frame information allows avoiding to evaluate the Boolean expression for distance calculations to primitive solids

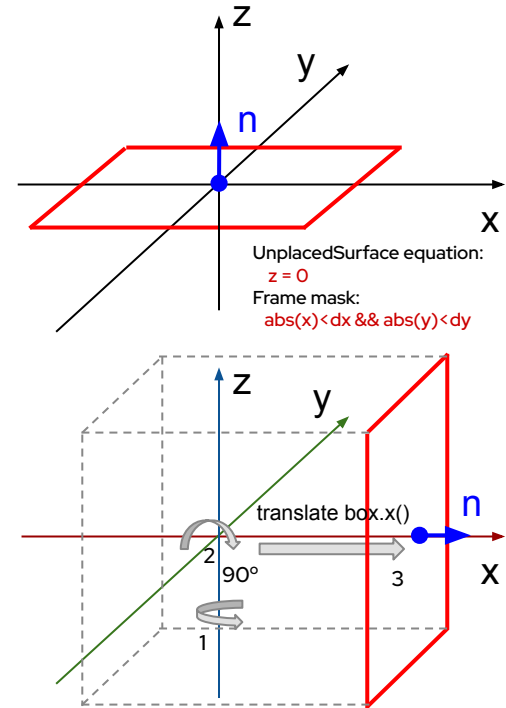


Objectives for the surface model

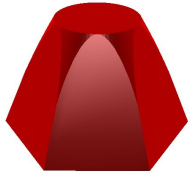
- ▶ Portable GPU-friendly header library
 - Algorithmic part independent on the backend, compilable with any native/portability compiler
 - Headers templated on the precision type to allow for a **single-precision mode**
 - Reduced set of simple surface/frame algorithms (vs. ~20 primitive solids now)
 - ▷ Reducing divergence and register usage on the GPU
- ▶ Automatic conversion from VecGeom transient solid model
 - Transparent creation of the data structures and copy to GPU
 - **Preserve awareness of the Geant containment feature as powerful optimization**
- ▶ Target: code simplification compared to the solid model
 - No virtual calls, no recursions, more work-balanced
 - Better device occupancy and kernel coherence

Conversion of existing solids

- ▶ Any solid surface can be made from predefined surface & frame types
 - 3D transformation + pre-defined surface/frame equations rather than transforming equations in the global reference, **for numerical stability**
- ▶ Generated framed surfaces behave like the tessellations in graphics
 - 'Real' hits allowing for faster distance reductions compare to the unbound approach
 - Except for the Boolean solids, which cannot compute the frames and have to be fully logically evaluated



Current status



Boolean



Trapezoid



Cone



Parallelepiped



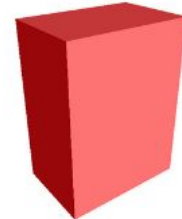
Polycone



Extruded



Trd



Box






Tube



Polyhedron

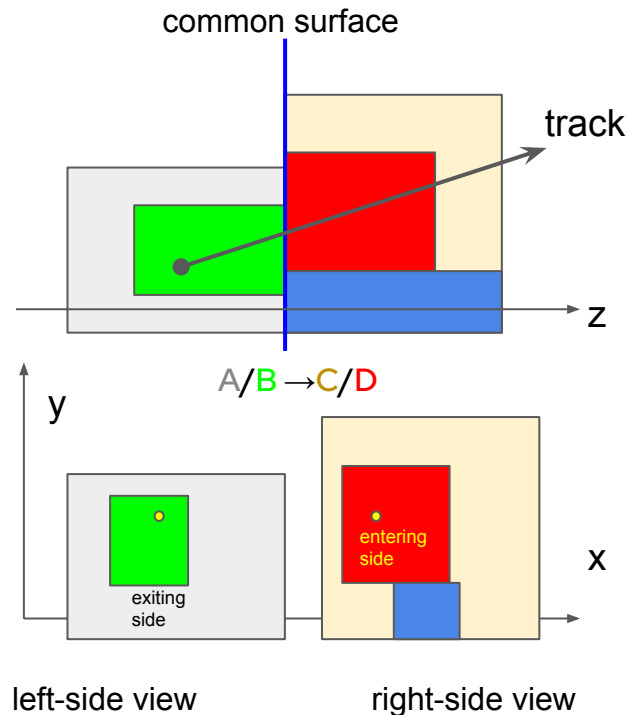
- ▶ **Locate** - needed only once per track
 - using CSG expression, reduction per solid
- ▶ **Relocate** - at each boundary crossing
 - frame search on common surface sides
- ▶ **Distance** - half-space + frame query (propagated point must be contained)
 - using hierarchic info constraining candidates
- ▶ **Safety** - combine half-space + frame distances

  Solids already added

 Solids to be added

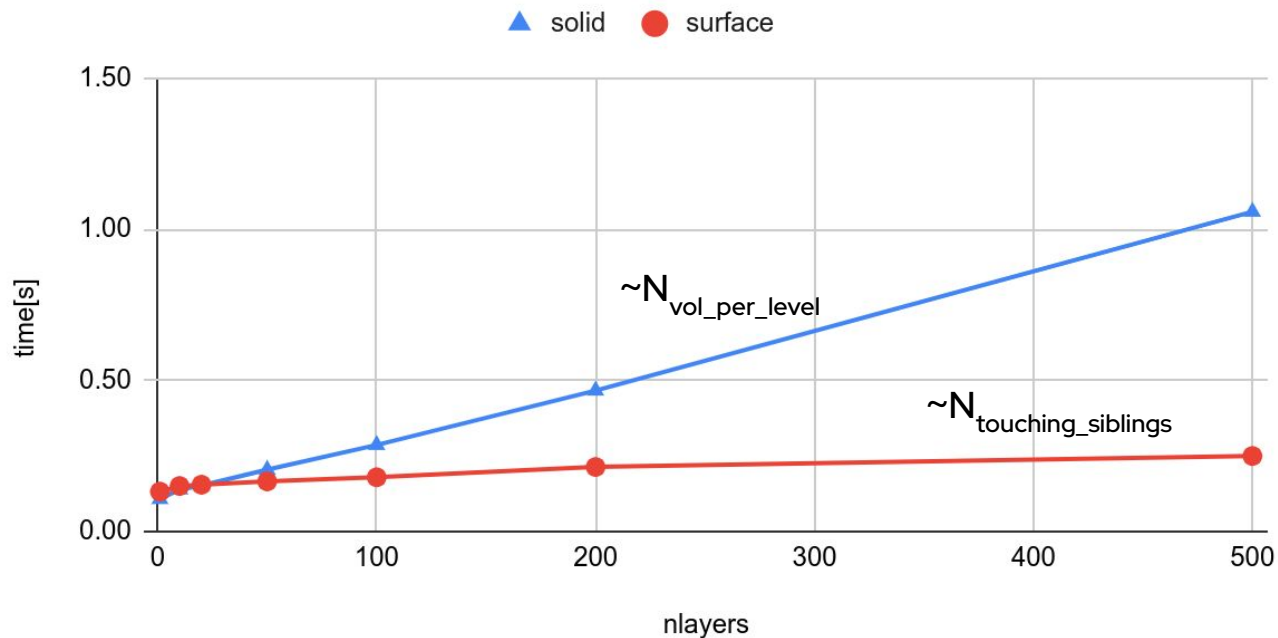
Non-recursive particle relocation

- ▶ In Geant volumes can share common surfaces
 - Define “*common surfaces*” as transition boundaries between volumes, pre-computed and deduplicated
 - Touching volumes contribute with frames on each side
- ▶ Locating the deepest frames hit on the surface sides allows finding the location after crossing volume boundaries
 - More efficient 2D linear search, involving only a **limited set** of neighbors and not all daughters of a volume
 - We can add 2D optimizers to mitigate the complexity at this level



Scaling for track relocation

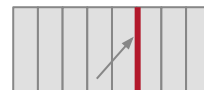
Multi-layered sampling calorimeter, distance computation & relocation



loop over objects at the same level



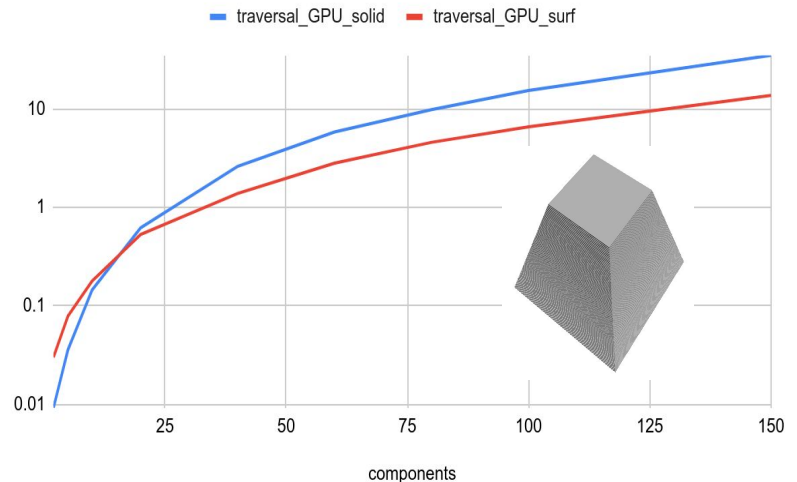
loop over frames on a common surface



Non-recursive Boolean implementation

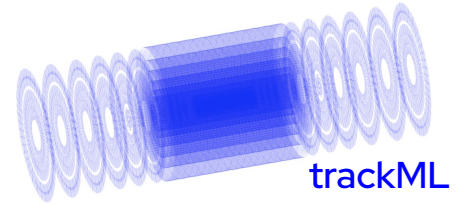
- ▶ Framed surfaces of Boolean solids can co-exist with non-Boolean ones on common surfaces
 - Linearized Boolean infix expression evaluation using Boolean algebra short-circuiting
 - Tested union of up to 150 layers of disks subtracting a box, more exhausts CUDA stack space for the solid approach
- ▶ Initial implementation scaling looking good
 - 2x slower for 5 components, 2x faster for 50 components on GPU

Traversal time for box tower



Ray-tracing example traversing a complex Boolean solid until exiting the setup

Preliminary performance



- ▶ Unit tests available for correctness checking against VecGeom solid model
 - Box, tube, trapezoid, polyhedron, Boolean solids
 - TestEm3 - a simple layered calorimeter made of box slabs
- ▶ Ray-tracing benchmark, working with generic GDML input (supported solids only)
 - Testing full navigation functionality on CPU and GPU
 - Validated & benchmarked against existing VecGeom solid navigators
- ▶ Results (compared to solid looping navigation) for trackML setup
 - Safety computation: ~2x slower on CPU, ~2x faster on GPU
 - Propagation + relocation: ~2x faster on CPU, ~6x faster on GPU
 - Memory: ~1.5 kByte per "touchable" volume

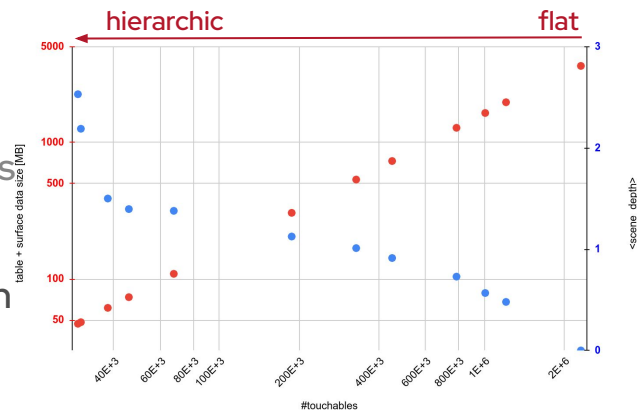
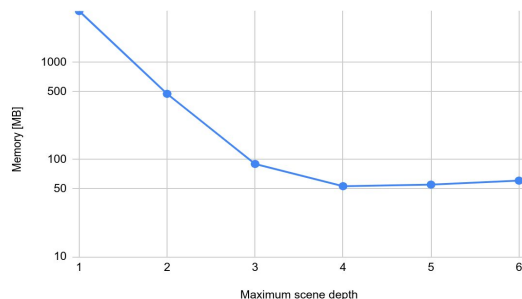
Memory mitigation

- ▶ Cannot afford to expand all touchables surfaces in memory
 - Including bounding box optimizations, we can easily reach $O(10)$ GBytes
- ▶ Adding support for the “scene” concept
 - Highly replicated volumes, kept once in memory, defining local “worlds”
 - A **state** in this approach can be represented by a tuple: (n_0, n_1, \dots, n_m)
 - ▷ Storing local surfaces and transformations
 - ▷ We can constrain the maximum depth `m` to a small value, with a huge impact on the memory footprint
 - ▷ Price to pay: conversion to the touchable frame done via successive transformations
- ▶ Now fully implemented and giving quite accurate estimates of the memory reduction
 - **Main challenge:** implementing scene-aware surface navigation - Done

Memory estimation for CMS Run2 setup

- ▶ The optimal choice of volumes to be kept as scenes is constrained
 - on the maximum scene depth, set at compilation time
 - ▷ plot above, a depth of 3-4 allows keeping the memory below 100 MB
 - on the minimum number of sub-hierarchy touchables
 - ▷ for a given maximum depth, this allows to fine-tune the average scene depth to minimize the number of extra run-time matrix conversions
- ▶ The impact on navigation performance
 - Seems negligible for simple setups, to be confirmed in complex ones

Geometry data size estimation in memory for cms_2018 for 1 million tracks in flight



Integration in AdePT GPU prototype

▶ Optional usage of the surface model in AdePT example

- No relevant changes needed other than triggering the model conversion and the navigator type

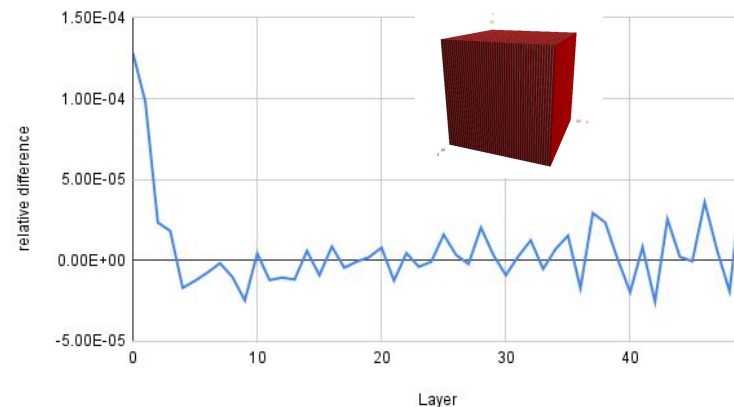
▶ Sampling calorimeter simulation

- block of Pb + LAr box layers (w/ constant Bz field)
- 10 GeV electrons shot towards the calorimeter along X axis

▶ Numerical divergence small and understood

- Boundary crossing relocation

EDEP relative difference TestEm3 100K electrons surface model vs. BVH (Bz = 1 Tesla)



	BVH	no BVH surf
no field	152s	156s
Bz=1T	194s	184s

Geometry to-do list

- ▶ Completion of the missing solid-to-surface conversion in CMS Run2 setup
 - polycone, cut tube, simple extruded
- ▶ Adding the adapted BVH acceleration for searches on common surface candidates
 - In future we may also need to reduce the search complexity within frames on common surfaces
- ▶ Validating the transport on CMS Run2 geometry (est. 2nd-3rd quarter 2024)
 - Measuring the impact on performance (including single-precision mode)
- ▶ Completing the model with the missing solids supported by Geant4
- ▶ Migration to version 2 of VecGeom (est. by the end of 2024)
 - Simplified CPU interfaces and portable surface-based GPU support

Test tools

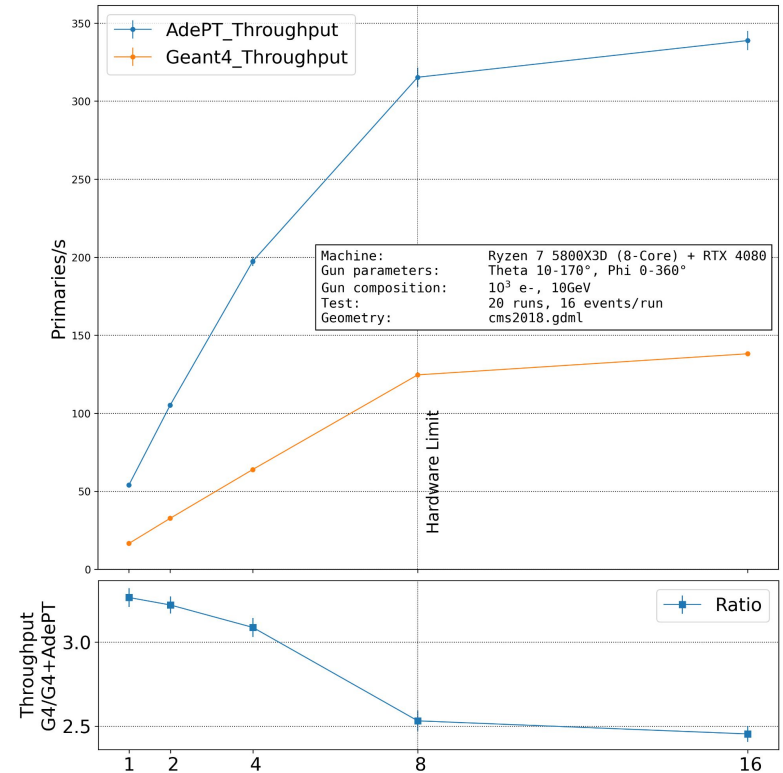
- ▶ Tools for **benchmarking and validation**
 - Timers, accumulators, export of results
 - Adapted to the needs found for AdePT
- ▶ Implemented in one of the integration examples
 - Easier extraction of in-detail data for further analysis
 - For example time spent simulating leptons inside the CMS ECAL
- ▶ Python scripts for automated testing and validation
 - JSON test configurations for easy setup and sharing
 - Automatic plotting of the results

Integration with fast-simulation Hooks

- ▶ AdePT transports particles in the ECAL region
- ▶ e^- , e^+ and gammas are buffered before triggering a shower
- ▶ Scoring is performed on the GPU when using AdePT
 - Simple scoring code as a placeholder for more complex options
 - Equivalent code on CPU and GPU

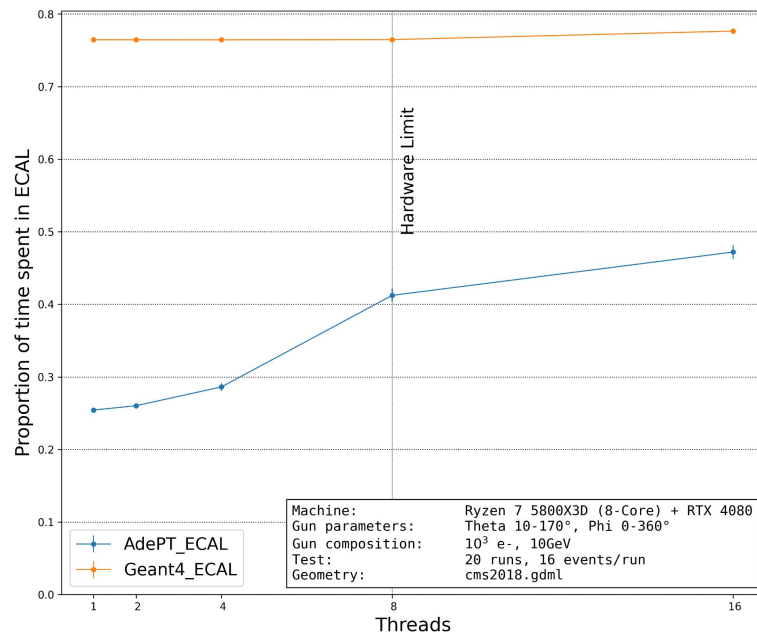
CMS Run2, electron bunches

- ▶ Not a realistic benchmark from event/track distribution perspective
 - ~75% of tracks transported in the ECAL
- ▶ Has the merit of showing how much the EM transport can be accelerated on per-detector basis



EM shower transport acceleration

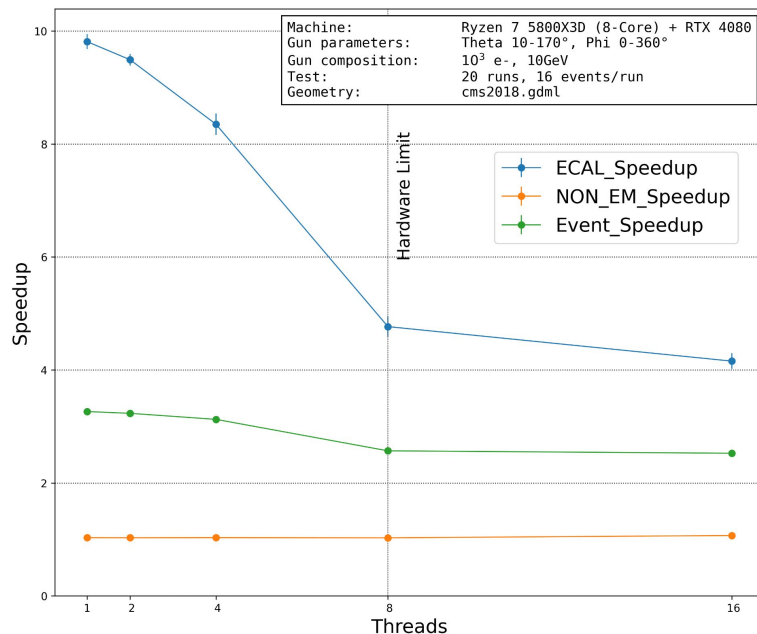
- ▶ Measure fraction of time spent simulating leptons in the ECAL
- ▶ Compare original fraction with Geant4 with improvement by AdePT



Proportion of time spent simulating leptons in the ECAL

EM shower transport acceleration

- ▶ Speedup of the ECAL simulation and overall event speedup
 - AdePT does not affect the rest of the simulation, 1:1 ratio in the time spent outside the ECAL
- ▶ Vary number of Geant4 worker threads
 - Decreasing AdePT speedup as the GPU becomes more saturated
- ▶ This integration approach works well for detectors where the EM transport is dominant



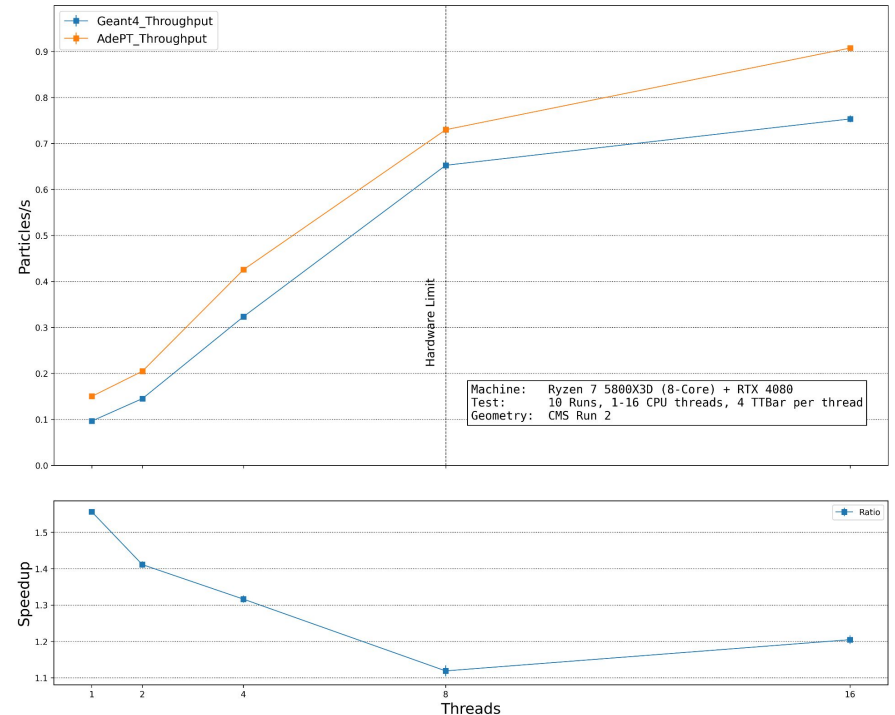
Speedup of the ECAL simulation and overall per-event speedup

Integration with a specialized Tracking Manager

- We no longer use a GPU Region
 - AdePT transports EM particles across the entire detector
 - Can also be used to enable the specialized transport only in specific regions
- Shows the current upper limits of performance
 - Eliminates overheads caused by sending back particles leaking outside the GPU Region
 - No realistic scoring performed on the GPU -> no overhead
- Is impacted more by the current GPU geometry inefficiency
 - VecGeom has to do the full detector in this case

CMS Run2, TTbar, full detector EM transport on the GPU, no field, Consumer grade GPU

- ▶ Shows a much better speedup when using TTbar events than the fast-simulation hooks integration
- ▶ The speedup increases when the CPU goes into hyperthreading, which shows that the GPU is not saturated and could take on more work

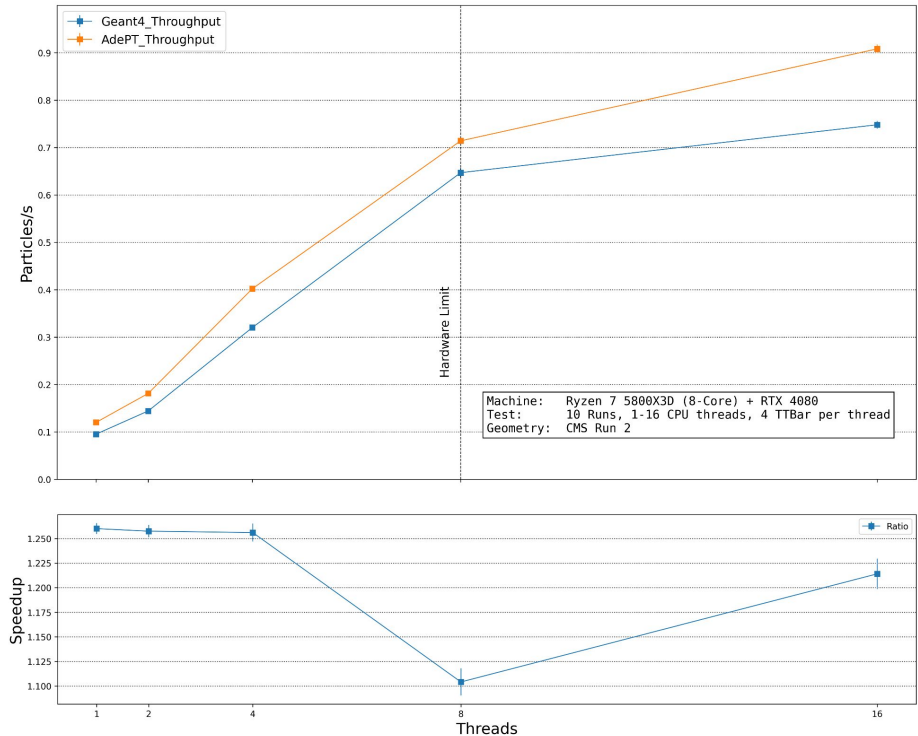


Integration reconstructing G4Step Information

- ▶ Uses the integration via Tracking Manager
- ▶ Different approach to scoring
 - We no longer have a GPU scoring code
 - Instead, we accumulate information about every step done in a sensitive volume on the GPU
 - This information is used to reconstruct the native G4Step and G4NavigationHistory on Host
- ▶ As a result, we are able to call arbitrary Sensitive Detector code without the need of a GPU port
 - This is the easiest-to-adopt integration mode for experiments to try out AdePT
 - Allowing to run the full simulation with their current scoring code

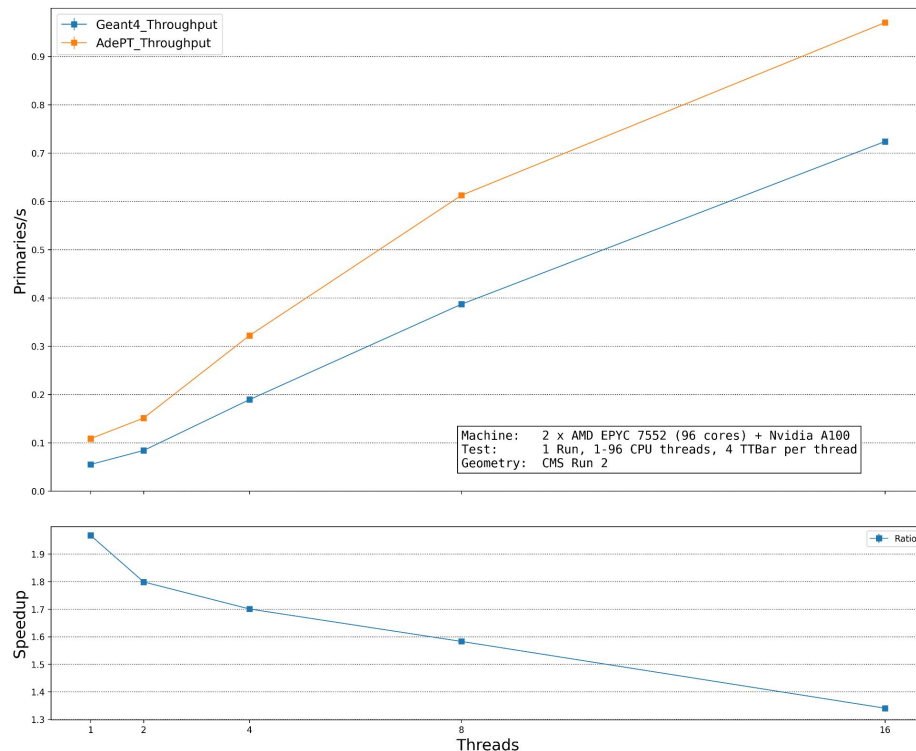
CMS Run2, TTbar, full detector EM transport on the GPU, Hits reconstructed on host, no field, Consumer grade GPU

- ▶ We see little to no overhead related to sending hit information back to the host
- ▶ The overhead is most visible when running with low thread counts, as a result of the reconstruction being done in sync with kernel launches



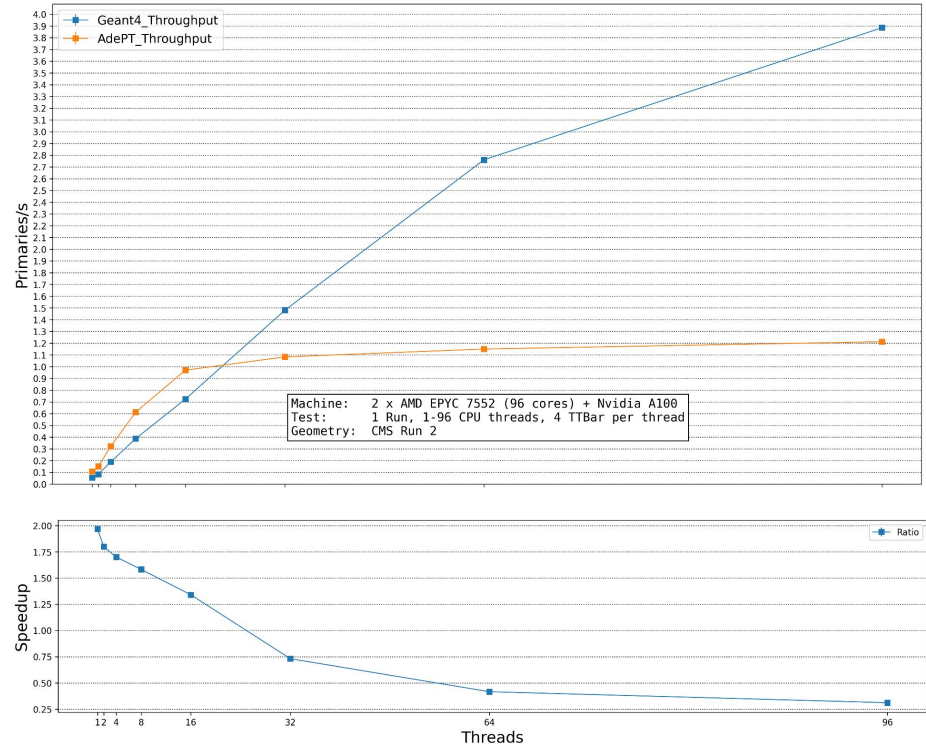
CMS Run2, TTbar, full detector EM transport on the GPU, no field, Nvidia A100, 1-16 Threads

- ▶ A faster GPU provides a better speedup than equivalent tests with consumer-level hardware



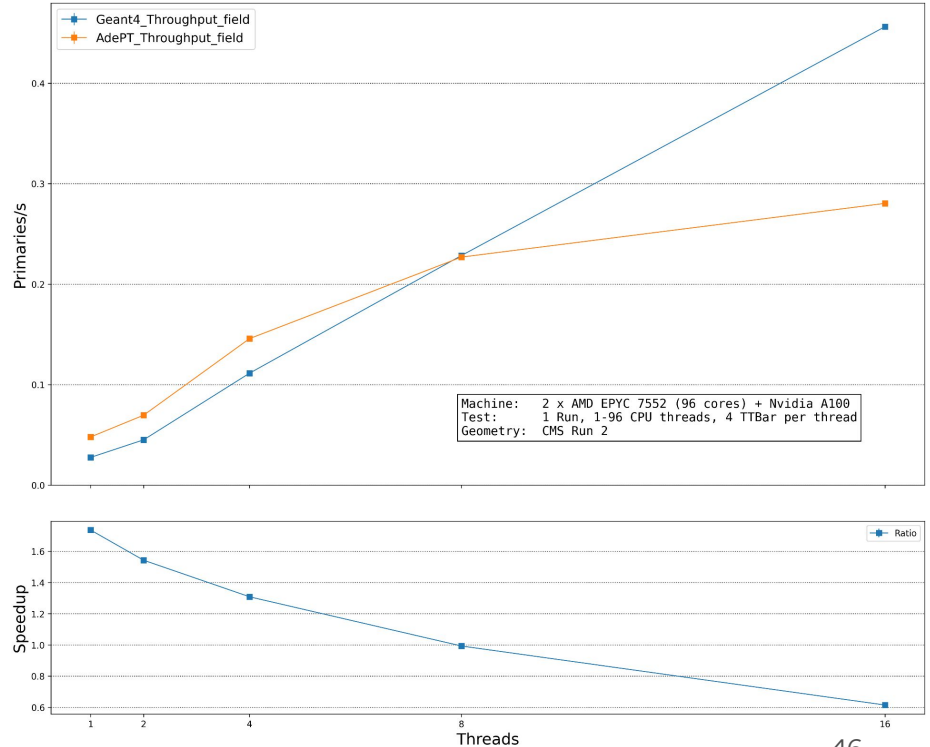
CMS Run2, TTbar, full detector EM transport on the GPU, no field, Nvidia A100, 1-96 Threads

- ▶ At a certain point the GPU becomes saturated. **Geometry** is a major factor in how early this happens
- ▶ Due to the current way of scheduling the kernel launches, this means that the GPU starts blocking the CPU threads
- ▶ More research into non-blocking scheduling strategies is **ongoing**



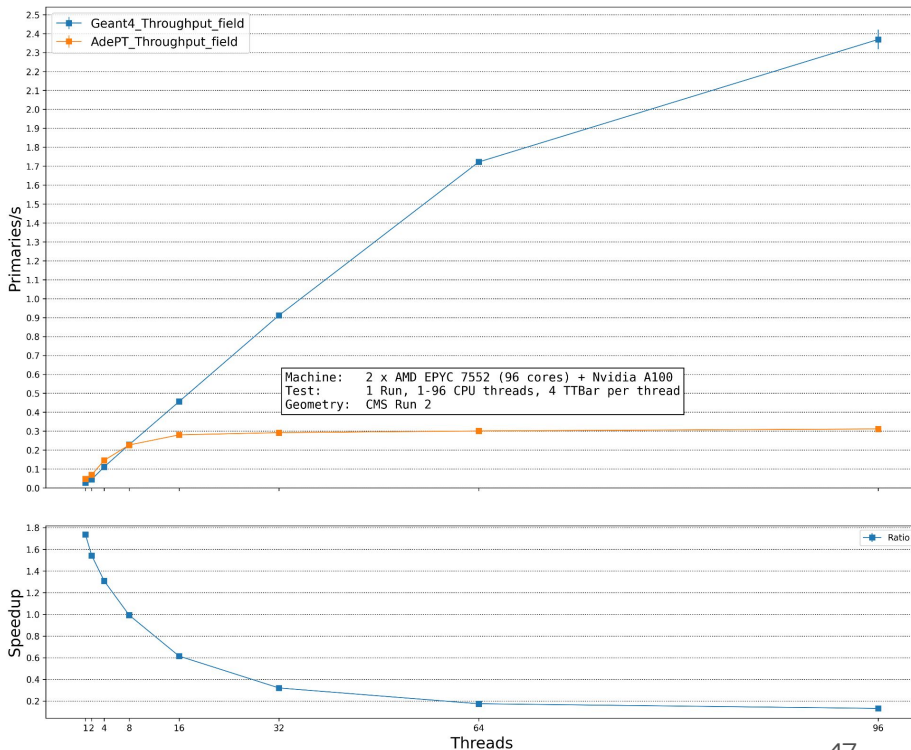
CMS Run2, TTbar, full detector EM transport on the GPU, constant 3.8T field, Nvidia A100, 1-16 Threads

- ▶ The speedup is smaller when running with a magnetic field. This is expected as the GPU is more penalized from this than the CPU
 - more geometry calls and divergence due to outliers
- ▶ There is still a significant performance gain while the device is not saturated



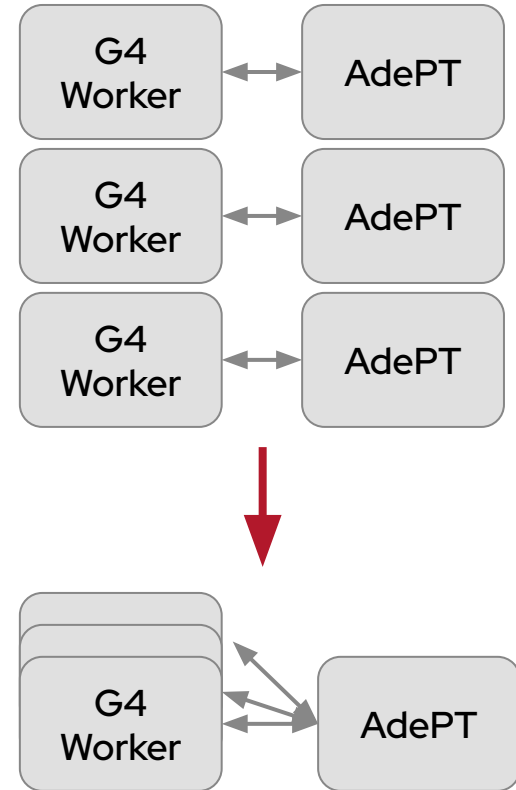
CMS Run2, TTbar, full detector EM transport on the GPU, constant 3.8T field, Nvidia A100, 1-96 Threads

- ▶ We observe similar results when running with more worker threads
- ▶ The GPU becomes saturated with lower worker counts. We still get a speedup before this happens
- ▶ An asynchronous scheduling strategy and an improved geometry could largely mitigate this issue and preserve the initial speedup when using more workers



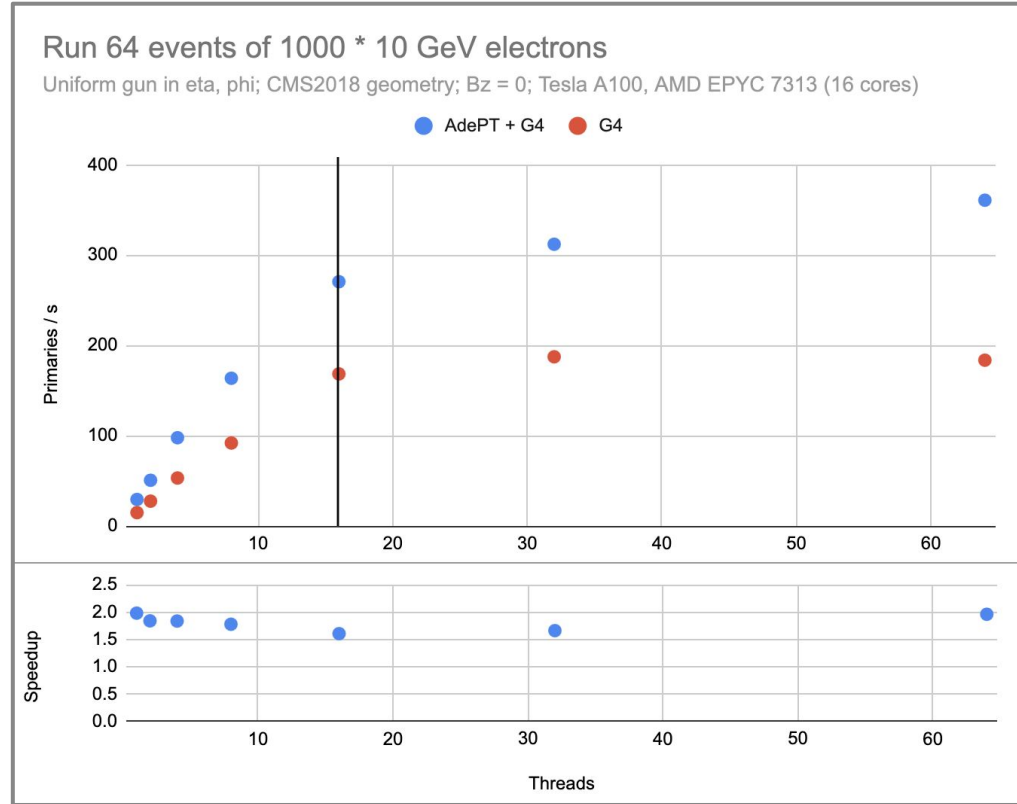
Asynchronous Shared AdePT

- ▶ **One** instance of AdePT runs in the background
- ▶ Transport loop runs continuously
- ▶ All G4 workers communicate with AdePT asynchronously
 - Host threads can continue with CPU work (e.g. Hadrons) while transport runs in the background
- ▶ Integrated to G4 as fast simulation model



Asynchronous Shared AdePT

- ▶ When ECAL is offloaded to GPU, can transport more events in parallel than CPU cores available
- ▶ Note the AdePT+G4 speed up with more threads **although CPU 2x and 4x overcommitted**
- ▶ Less device memory usage
→ **can transport more tracks concurrently**



Mitigation of GPU Saturation

▶ New VecGeom Surface Model

- A reduced set of surface algorithms compared to the current solid-based implementation will reduce branching on the GPU
- We expect a significant improvement in device occupancy due to lower register usage, which can increase the amount of work the GPU can take before becoming saturated

▶ Non-Blocking Scheduling strategies

- The current scheduling strategy blocks the CPU threads while the GPU is transporting particles
- This is a less-visible issue while the device is not saturated
- When the GPU becomes overloaded it slows down the CPU as well
- The asynchronous AdePT prototype shows a non-exclusive scheduling strategy which makes better use of resources
 - ▷ Tasking-based approaches will be also looked-upon

Propagation in magnetic field

- ▶ Uniform magnetic field currently implemented using a helix
- ▶ Runge-Kutta integration method is currently being tested and validated
- ▶ An initial benchmark comparing Helix and Runge-Kutta was done on the TestEm3 setup for a single-threaded application
 - Further testing is underway with more complex setups and multi-threaded applications. Optimization will follow.

	B= 1.0 T	B= 3.8 T
helix	595	671.6
Runge-Kutta	700	985.5

Integration with experiments

- ▶ Making AdePT and its dependencies (geometry and physics) easier to integrate in experiment frameworks
 - Solve **libraries incompatibilities, linking problems**, etc
 - ▷ Already solved in case of CMSSW
 - Improving the way AdePT can be used externally
 - ▷ Each experiment has its own way of building the geometry and configuration
- ▶ Testing AdePT in **more complex setups**
 - Geometry, particles input/output, etc
- ▶ Studying the impact of the current scoring approach with **realistic sensitive detectors**
 - Delivering Geant4 step information seems to mitigate scoring problems
 - ▷ To be confirmed with realistic use cases

Ongoing and future work

- ▶ GPU geometry model
 - Taking most of the development effort
 - Larger collaboration would accelerate reaching the common goals
- ▶ Improving the CPU-GPU parallelism model
 - More efficient CPU utilization while the GPU is saturated
 - Optimize alternative GPU dispatching approaches: sub-tasking, single-threaded
- ▶ Validation and optimization for **non-constant field** implementation
 - Extending the current RK implementation to more advanced examples
- ▶ Integration with **experiment frameworks** and validation
 - Started, hoping to get more momentum soon

Achievements

- ▶ **Transport for EM particles working on GPUs** for LHC-complexity geometries ✓
 - **Excellent physics agreement** within statistical fluctuation ✓
 - **Reproducibility** of the simulation achieved ✓
- ▶ **Full integration with Geant4 applications** ✓
 - Fast simulation approach
 - Custom tracking
 - Reusing existing sensitive detector implementations
 - ▶ Possible to plug AdePT into existing Geant4 applications with minimal extra code ✓

Summary

- ▶ Achieved the initial goals of the R&D
- ▶ Integration with experiments is ongoing and growing activity
 - Study and optimisation of AdePT performance within experiments framework
- ▶ The CPU-GPU workflow currently implemented in AdePT can boost performance in configurations combining equivalent CPU and GPU power
 - Further performance gain potential actively explored
 - The new surface model expected to largely boost performance by removing the bottleneck related to the current geometry implementation
 - ▷ To be validated in the first part of next year for complex setups

AdePT developers and contributors

Guilherme Amadio, CERN

John Apostolakis, CERN

Predrag Buncic, CERN

Gabriele Cosmo, CERN

Dusan Cvijetic, EPFL

Daniel Dosaru, EPFL

Andrei Gheata, CERN

Juan Gonzalez, CERN

Bernhard Manfred Gruber, CERN

Stephan Hageboeck, CERN

Jonas Hahnfeld, CERN

Mark Hodgkinson, University of Sheffield

Vladimir Ivantchenko, CERN

Ben Morgan, University of Warwick

Mihaly Novak, CERN

Antonio Petre, University of Bucharest

Witold Pokorski, CERN

Alberto Ribon, CERN

Eduard George Stan, ISS Bucharest

Graeme Stewart, CERN

Pere Mato Vila, CERN